Modelling techniques for spatial information using free and open source tools

Patrick Browne, School of Computing**,** Dublin Institute of Technology **,**Kevin Street, Dublin 8, Ireland.

Mike Jackson, School of Computing and Information, University of Central England, England.

(references in main paper)

Outline

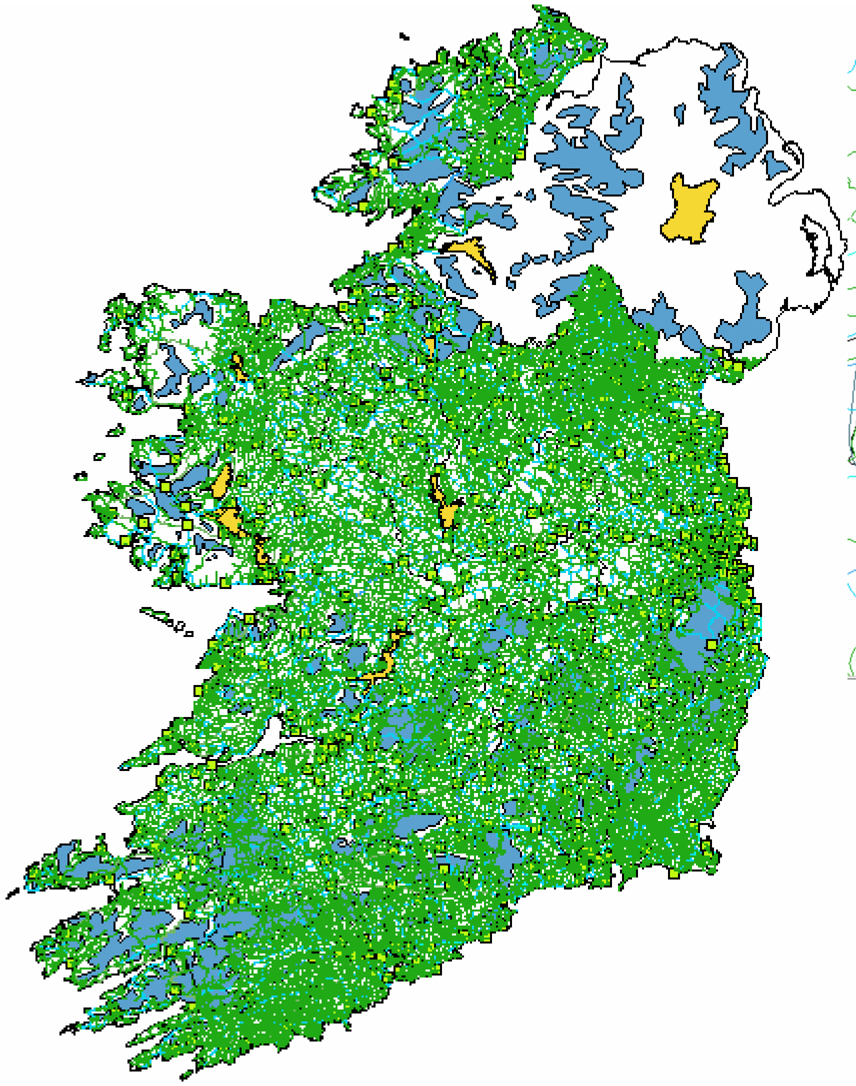
- Motivation. Why model?
- Brief outline of tools
  - ArgoCASEGEO
  - USE
  - CafeOBJ
- Simple maps
- A software representation of the maps
- Description of tools and how they represent classes, attributes, associations and spatial data and spatial relationships.
- Commercial system
- Conclusion

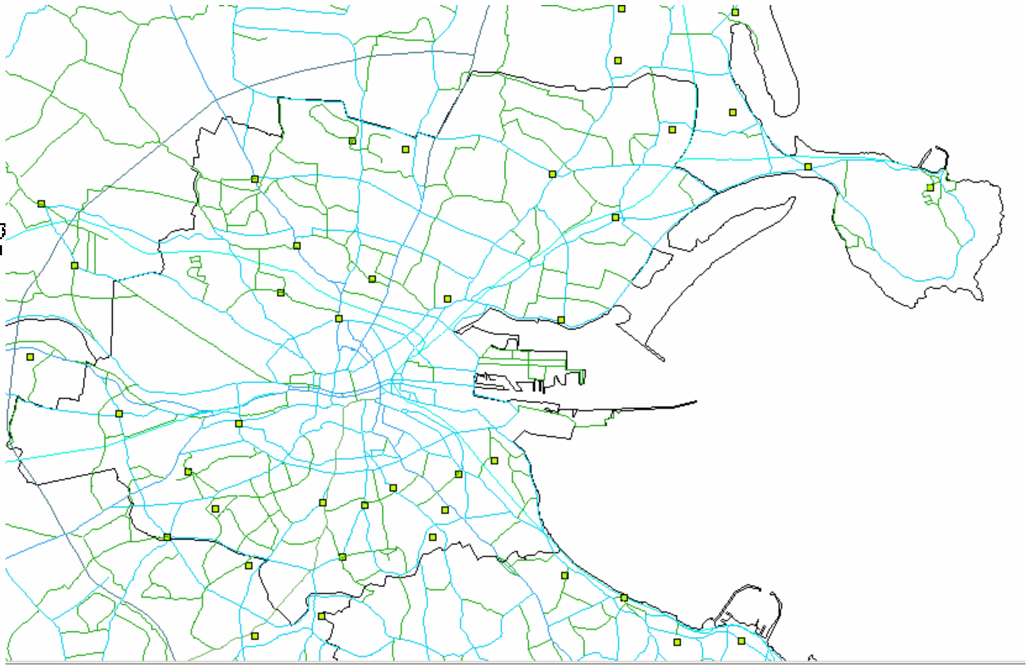
What is a GIS?

- Bittner and Frank 1999, state that
- “*GIS is an implementation of formal theories of geographic space*”.
- This fits in with our view that formal specifications are of central importance in GIS.

Nature of geo-data and geo-processes

- Geographic information is complex:
  - It is heterogeneous, metric, topological, thematic, and time varying.
  - Needs to be stored, queried, and updated, which entails the specification of a database schema.
- Modelling can help develop applications, standards, systems, and contributes to our understanding of complexity.





Geo-data and processes need to be designed using the appropriate techniques.

Mr. Patrick Browne, School of Computing
Dublin Institute of Technology

Models

- We can view a model
  - as a toy (a prototype)
  - as an ideal (obeying axioms)
- Models are required so that we can better understand a domain of interest and the system we are developing to operate in that domain.
- Where to formalize?
  - At specification level
  - At program level
- An important model is the` computation specified by a program.

Fundamental importance of a model

- In computing: *no information or computation without representation and representations describe models.*
- Cantwell Smith (Cantwell-Smith 1985) says
- “*Models are ubiquitous not only in computer science but also in human thinking and language ; their very familiarity makes them hard to appreciate .*”
- “*To build a model is to conceive of the world in a certain delimited way*.”
- “*A program is an explicit description of a model contained inside the computer*”
- Do we model explicitly or implicitly?

Why Model?

- Geo-data is very expensive to collect and maintain, hence in order to maximize potential usage its *representation* needs to be in canonical form capable of serving a wide range of applications.
- Having a ‘formal semantic’ or sound model reduces the problems of interoperability.

Why Model?

- Models help us to visualize a system as it is or as we want it to be.
- Models permit us to specify the structure or behaviour of a system.
- Models give us a template that guides us in constructing a system.
- Models document the decisions we have made.
- Emergence of the Model Driven Architecture (MDA).
  - intellectual value lies in the model.

Who Models?

- Standards Bodies: OGC, ISO, local authorities, national and international bodies.
- The geo-researcher.
- Mapping Agencies.
- User Communities.
- Academic and commercial GIS builders.

Levels of Modelling

Ontology engineering

Conceptual data modelling

Database Schemas

Model refinement

Algorithm design

Program design

Modelling access methods

Levels of Modelling

- Gruber describes ontoogies as: “formal, explicit specification of a shared conceptualization”.
- Ontologies provide generic and task-independent view of the world. They focus on more abstract rules such as totality, rigidity and identity. They are formally agreed logical theories which can be shared between many different applications. Designing an ontology involves “considering the subjects separately from the problems or tasks that may arise or are relevant for the subject”
- Conceptual data modelling (CDM) specifies the structure and integrity if data sets. Building a data model for an organisation usually depends on the specific needs and tasks that have to be performed within this organisation. Data modelling semantics are generally created as without any formal agreement between user communities.

The value of a formal model

- The main advantages of a formal model are expressiveness, precision, composition, and formal models match the type of mathematical components common in GIS.
- Currently geo-standards are expressed using a semi-formal approach, namely the UML.
- Leading geo-researcher Frank and Kuhn state that “*GIS is stretching current software design methods to the break point*”.
- Though we focus on formal aspects, in practice “sliding scale of formality” may be more appropriate (structured English, UML/OCL, type checking, algebraic specification, refinement and proofs).

Why an executable model?

- A formal and expressive language is needed to precisely and concisely define systems. This could be a high level formal specification or at a lower level of abstraction computer program. However at some point English requirement must be formalized, by either a designer or a programmer. Early validation can be provided by an executable specification which gives immediate feedback of the design decisions encoded in the specifications

Approaches and Tools

- Objected Oriented:
  - ArgoCASEGEO + USE
- Algebraic and logic based
  - CafeOBJ

Object Oriented Tools

- Diagrams: ArgoCASEGEO (Lisboa, Sodré et al. 2004) extends the Unified Modelling Language (UML). It adds spatial and temporal types to the basic UML.
- Executable Specification: USE is an Object Constraint Language (OCL) tool (Richters 2006). Constraints can be written and tested against instances.
- Together these tools represent the object oriented paradigm.

Algebraic Specifications

- CafeOBJ is an algebraic executable specification language called CafeOBJ (Diaconescu, 1999).
- CafeOBJ in many ways is a much more expressive language than UML/OCL, it is both a specification language and a functional programming language. Models are represented by sorts (e.g. sets) and operations (e.g. functions) on those sorts. The operations are represented as mathematical equations. No graphics.

Simple Maps

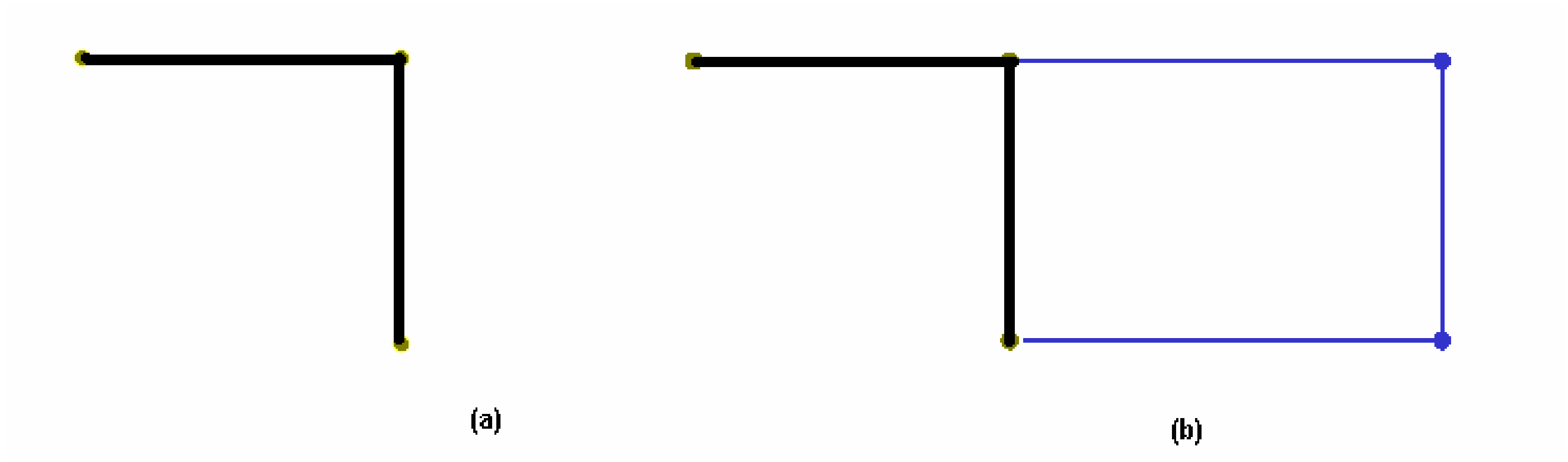


Figure 1(a) and 1(b)

Black = Road

Blue = Administrative Region

Description of Map

- If we first consider the lines as pure ‘geometry’ then, individual lines have the metric property of length. The lines can be composed to form a path. Paths with some special properties (i.e. cycles) can form boundaries of regions. We can reason about paths and boundaries using graph theory. Now consider Figure 1(a) as a topographic object we will consider the lines as road section that can be traversed as a path which also has a length (the length of the sum of its line segments). Hence we are combining in a very simple and informal way the metric, graph theoretic, and thematic information about a topographic object that we wish to consider as a road section.

Another Map

- A thematic object has a different meaning compared to its component geometric objects. Consider the `beside` relationship Road 1 is beside Field 1 and Field 2 and forms part of their boundaries so points 1 and 2 must exist. Roads do not split regions, so the road has no explicit `beside` relationship with Administrative Region 1 and does not need points 1 and 2.

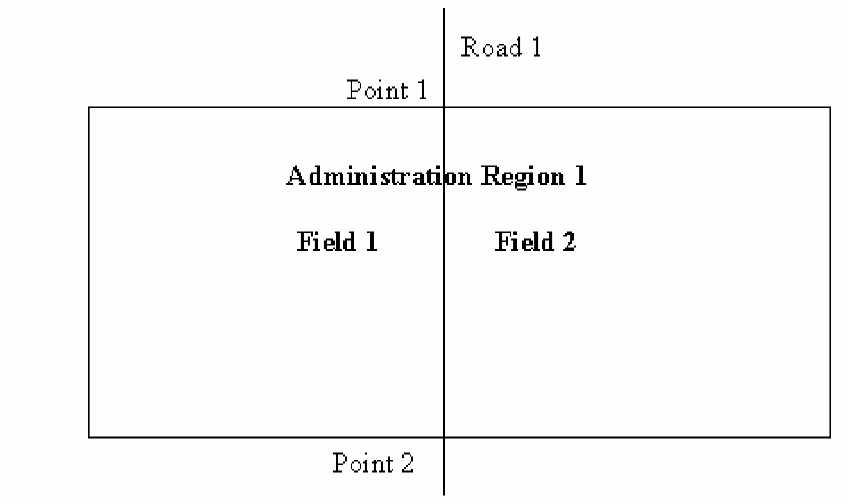


Figure 2 Map 2

Classes and Associations Map 1(a) in USE

Classes	**Associations**

```
model BasicSpatial
class Point
attributes
x : Integer
y : Integer
end
class Line
end
class Path
end
class Polygon
end
```
	```
	-- a line must have two points
	-- a point can be in zero or many lines.
	association
	Line_Point between
	    Line[0..*]
	    Point[2] ordered
	end
	association Polygon_Path between
	    Polygon[0..*]
	    Path [1..*] ordered
	end
	association
	Path_Line between
	    Path[0..*]
	    Line[1..*] ordered
	end
	```

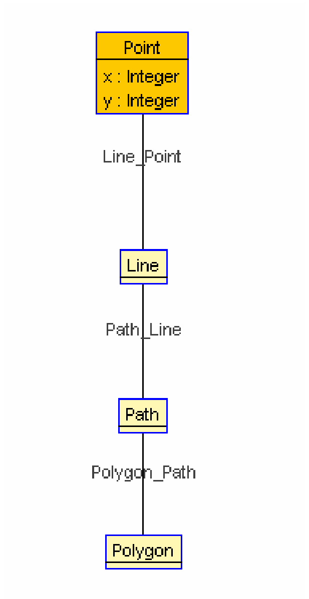


Figure 3 shows the class diagram generated by the USE tool.

Object and Association Creation in USE

```
-- create objects for figure 1(a)
!create p1 : Point
!create p2 : Point
!create p3 : Point
!create l1 : Line
!create l2 : Line
!insert (l1,p1) into Line_Point
!insert (l1,p2) into Line_Point
!insert (l2,p2) into Line_Point
!insert (l2,p3) into Line_Point
```

```
-- create extra for figure 1(b)
!create p4 : Point
!create p5 : Point
!create l3 : Line
!create l4 : Line
!create l5 : Line
!insert (l3,p2) into Line_Point
!insert (l3,p4) into Line_Point
!insert (l4,p4) into Line_Point
!insert (l4,p5) into Line_Point
!insert (l5,p5) into Line_Point
!insert (l5,p3) into Line_Point
```

USE has three languages 1) a Java like class definition language 2) OCL and 3) a command languages for creating objects and associations. The USE `model` is somewhat like a UML package, but only one model at a time can be opened in the USE tool.

Above are the commands needed to construct an instance of the maps in Figures 1(a) and 1(b). Figure 5 shows object diagram (i.e. an instance). It is also possible to construct interaction diagrams for say an update operation with the USE tool.

Object Diagram Map 1(b) in USE

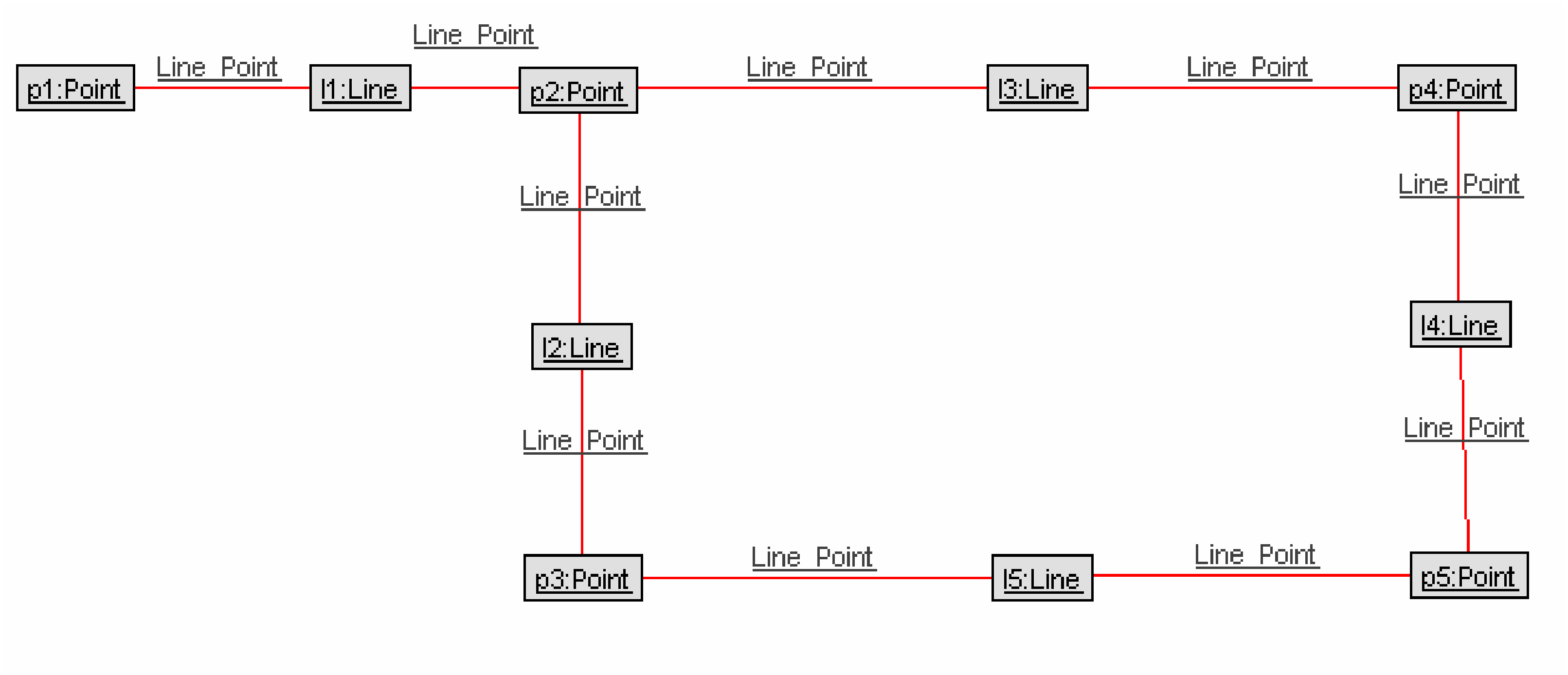
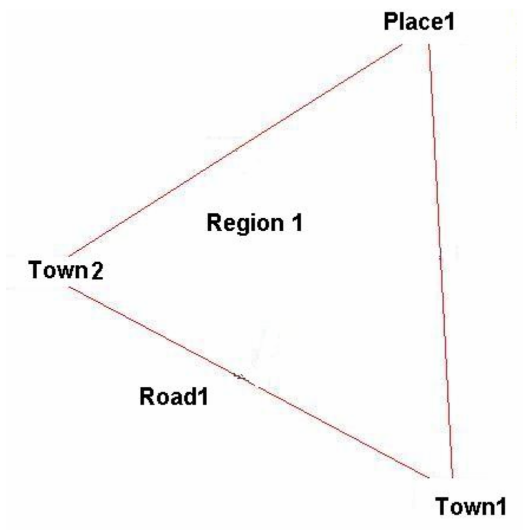
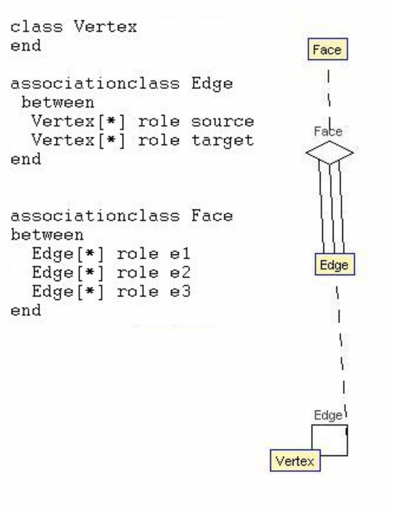


Figure 4 shows object diagram (i.e. an instance). It is also possible to construct interaction diagrams for say an update operation with the USE tool.

Using Association Classes in USE





```
class Vertex
end

associationclass Edge
 between
  Vertex[*] role source
  Vertex[*] role target
end


associationclass Face
between
  Edge[*] role e1
  Edge[*] role e2
  Edge[*] role e3
end
```

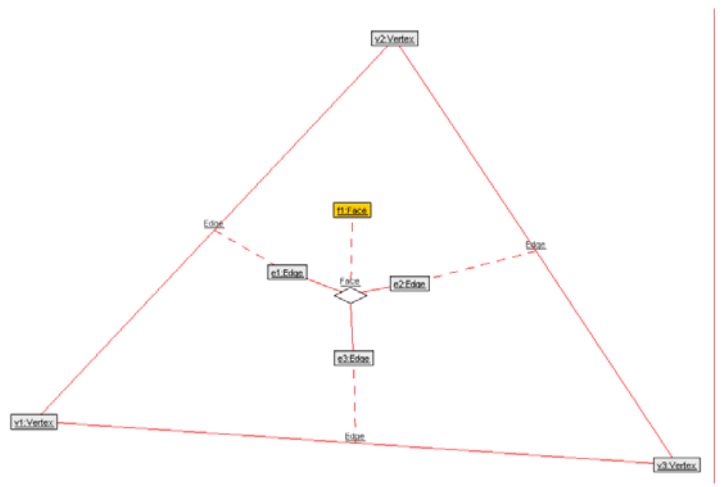


Figure 5. A map class definition and instance diagram representing the topology of the map.

OCL Constraints

- Constraints are rules we want to be true in our specification and ultimately in our system. For example, multiplicities that establish how points, lines and polygons may be related. USE does not allow duplicate instances of an association. Hence in our geometry there are no duplicate lines between points, if we tried to make the same point a start and end of the same line USE would tell us that the association already existed. This constraint could be expressed explicitly as:

```
context Line DistinctStartEnds inv:
-- constrains the two ends of a Line to be different
point -> forAll(p1,p2 | p1 <> p2)
```

- If we entered the command
- `check DistinctStartEnds`
- The USE tool would examine the current instance diagram and report whether or not the constraint had been violated. This is a static constrains (called an invariant) which must be true for all instances of a class. USE can use more dynamic constraints on operations called pre-conditions and post-conditions.

OCL Queries

- Here is an example of a simple query that returns any roads named Main Street.

```
-- using a variable put two roads into a set.
!let roads = Set{r1,r2}
-- query to find all main streets
?roads ->select(r : Road | r.name = 'Main
  Street')
```

- We can define an operation in the Polygon class that finds triangles as follows:

```
getTriangles() : Set(Polygon)
 = Polygon.allInstances->
       select(p : Polygon | p.line->size = 3)
```

Logic in USE

- **Logic**
- The querying and general assertion mechanism is based on first order predicate calculus. Querying can use a variety of techniques such as navigation via associations (or attributes), quantified statements (forAll, exists). OCL also has collections such as sets and sequences which can to an extent simulate a database. The limitations of OCL (and hence USE) as a database are described by Mandel and Cengarle {Mandel, 1999}. OCL uses logic but is not based on logic.
- **Other features.**
- USE has a monitor and generator features which allow validation and testing of a software implementation against the UML/OCL model. The USE tool can model Aspect Oriented Programming (AOP) and facilitates mappings between different abstraction levels of software. In order to keep things simple we have not presented any details on state and behavior (attribute values and operations that change those values). USE does support such concepts and allows users to construct snapshots of a system and examine in detail the changes that may occur between snapshots.

Summary of USE

- Use allows the user to construct executable specification, which can be refined to the level of a simple prototype. The prototypes are very high level and are not committed to any particular programming language or database system. However USE does follow the object oriented (OO) paradigm and hence favours an OO implementation. There are tools that do code generation, converting high-level constrains in OCL to Java programs {Dresden, 2006}. USE allows the analyst to explore ideas, to construct models, and prototypes without the labour of programming or setting up a database. It is particularly useful for exploratory work where the available software does not support a required feature (e.g. experimenting with various temporal theories for GIS). We found that the interactive nature of tools like USE is an important factor in clarifying and constructing conceptual models (e.g. studying update).

ArgoUML

- ArgoUML, a free open source UML modeling tool. It is compatible with UML 1.3 and Java. It can read XMI V1.1 and Java class definitions and it can create UML class diagrams. Fig 6 represents an approximation of the OpenGIS Simple Features Specification for CORBA (Department of Geography, University of Bonn). The diagram was generates semi-automatically by ArgoUML. This example shows how ArgoUML can be useful in understanding a complex class hierarchy.

ArgoUML

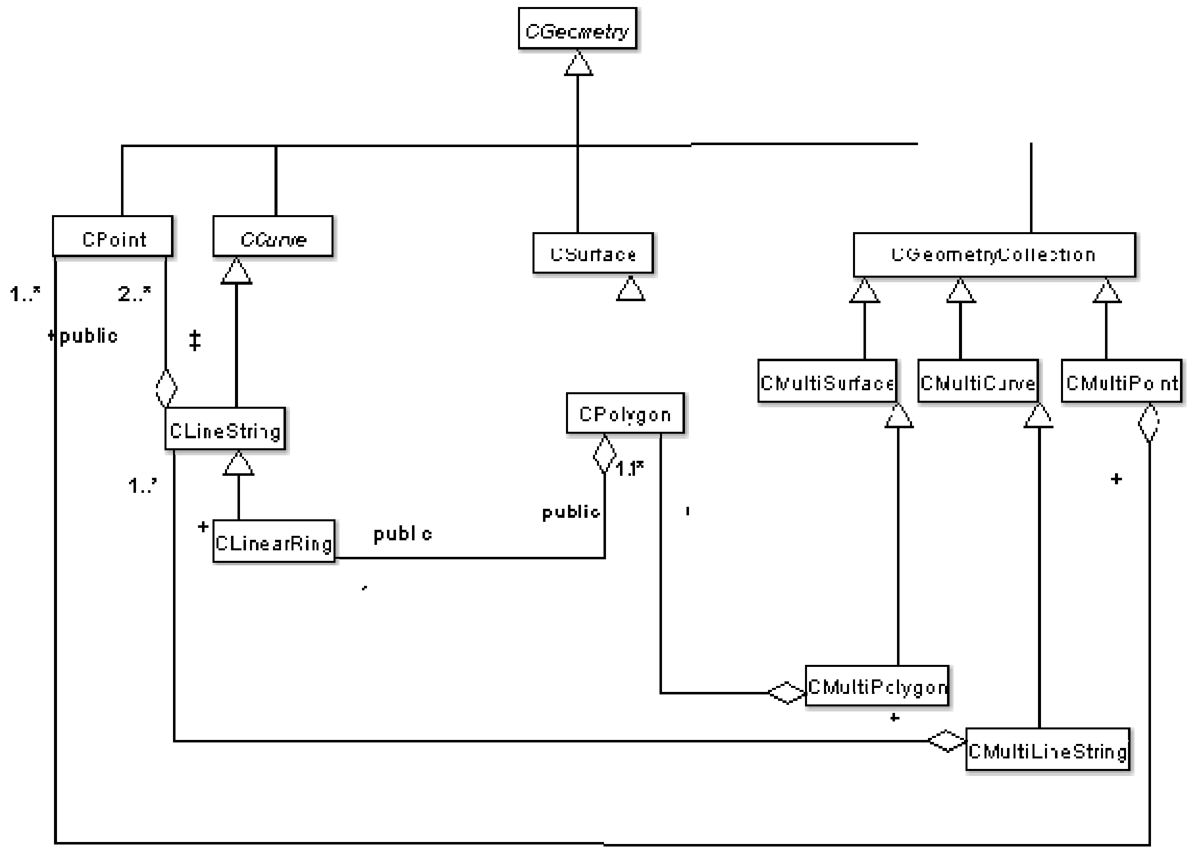


Figure 6. OGC Simple Features reversed engineered from JAVA code

ArgoUML

- Models can be saved as ArgoUML projects, XMI interchange format, or graphic files. ArgoUML uses XMI V1.1 and Poseidon uses XMI V1.2, ArgoUML. Importing ArgoUML models into Poseidon is fairly straightforward. You unzip the ArgoUML project file (.zargo) and extract the XMI file, which is loaded into Posiedon via the project menu. Going in the other direction can cause problems. Meta Integration (MetaIntegration, 2006) provides a commercial solution to XMI conversion, which permits you to import Rational Rose and Poseidon models into ArgoUML.

ArgoUML

- ArgoUML does not supports association classes or object diagrams (only in a limited way via component diagrams).
- ArgoUML parses OCL statements and maintains the OCL code for a particular model. To make full use of OCL it needs to be brought into (via cut and paste) USE tool where objects can be instantiated and constraints can be executed.
- OCL does not support specialized spatial or temporal constraints.
- These two pieces of open source software work very well together, ArgoUML providing the diagramming facilities and USE provide the more formal expression of rules and the ability to check those rules.
- Using standard UML the user can represent spatial objects, but they are very much on their own unless they have spatial libraries.

ArgoCASEGEO

- The general modelling facilities of ArgoUML have been extended to by Lisboa et al to handle the modelling of geo-data. The formal model behind ArgoCASEGEO is the UML-GeoFrame. This is an extension to the UML which adds a spatial object view (points, lines and polygons) and a spatial field view (grids and TIN) to the basic object modeling provided by the UML.
- Figure 7 from Lisboa et al describes the architecture of ArgoCASEGEO. It uses the UML-GeoFrame conceptual model rather than the more standard OGC/ISO standard. It outputs XMI, ESRI's ArcView 3.2 shape files, Intergraph's GIS environment GeoMedia, OGC and a GIS class library called TerraLib. ArgoCASEGEO uses transformation rules to map from the UML-GeoFrame to the various output formats. The shape files can be used as 'empty layers' in ESRI products such as ArcExplorer or open source products such as OpenMap. ArgoCASEGeo will transform the UML-GeoFrame version of Point, Line and Polygon to the ESRI format.

ArgoCASEGEO

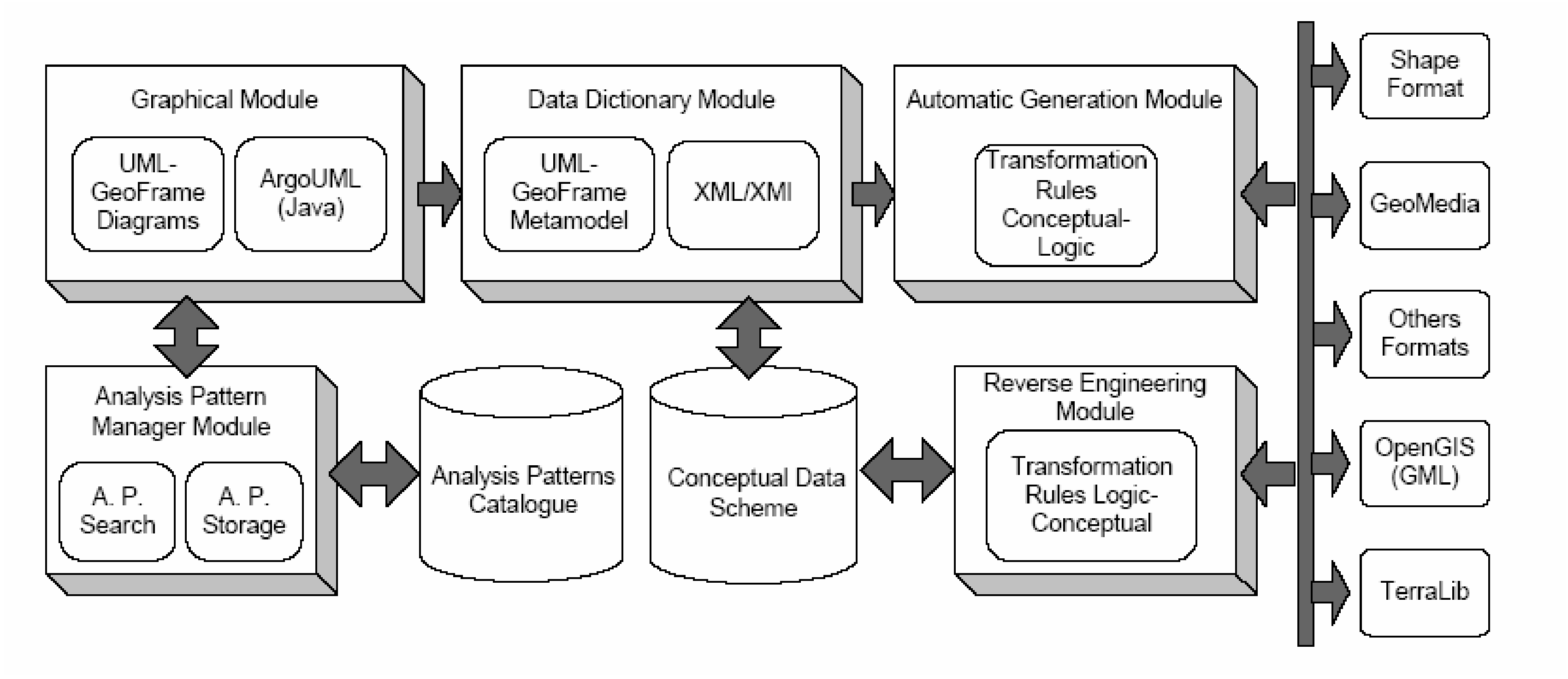
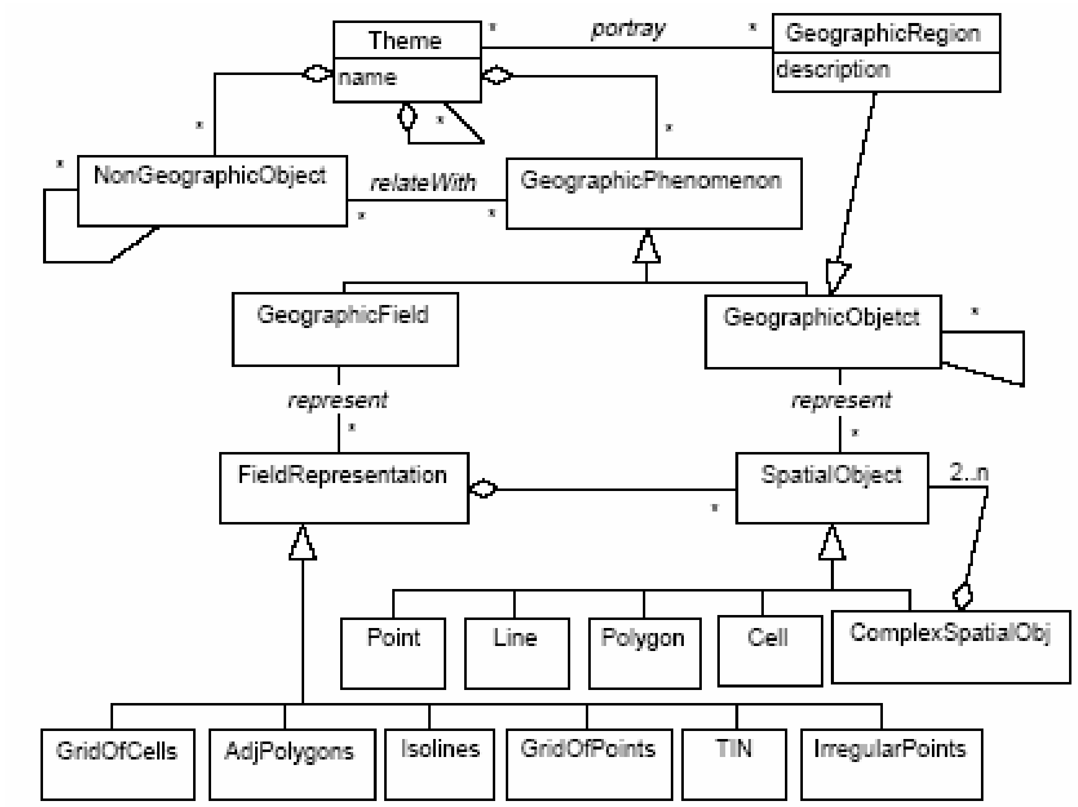
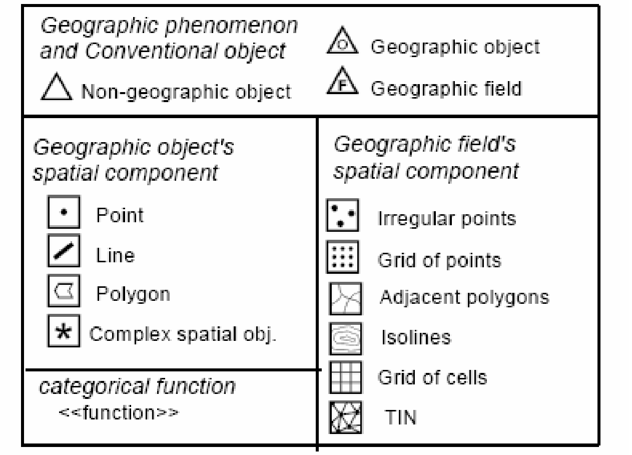


Figure 7 The ArgoCASEGEO Tool Architecture

UML-GeoFrame

The conceptual framework used by ArgoCASEGEO is UML-GeoFrame





Class diagram and

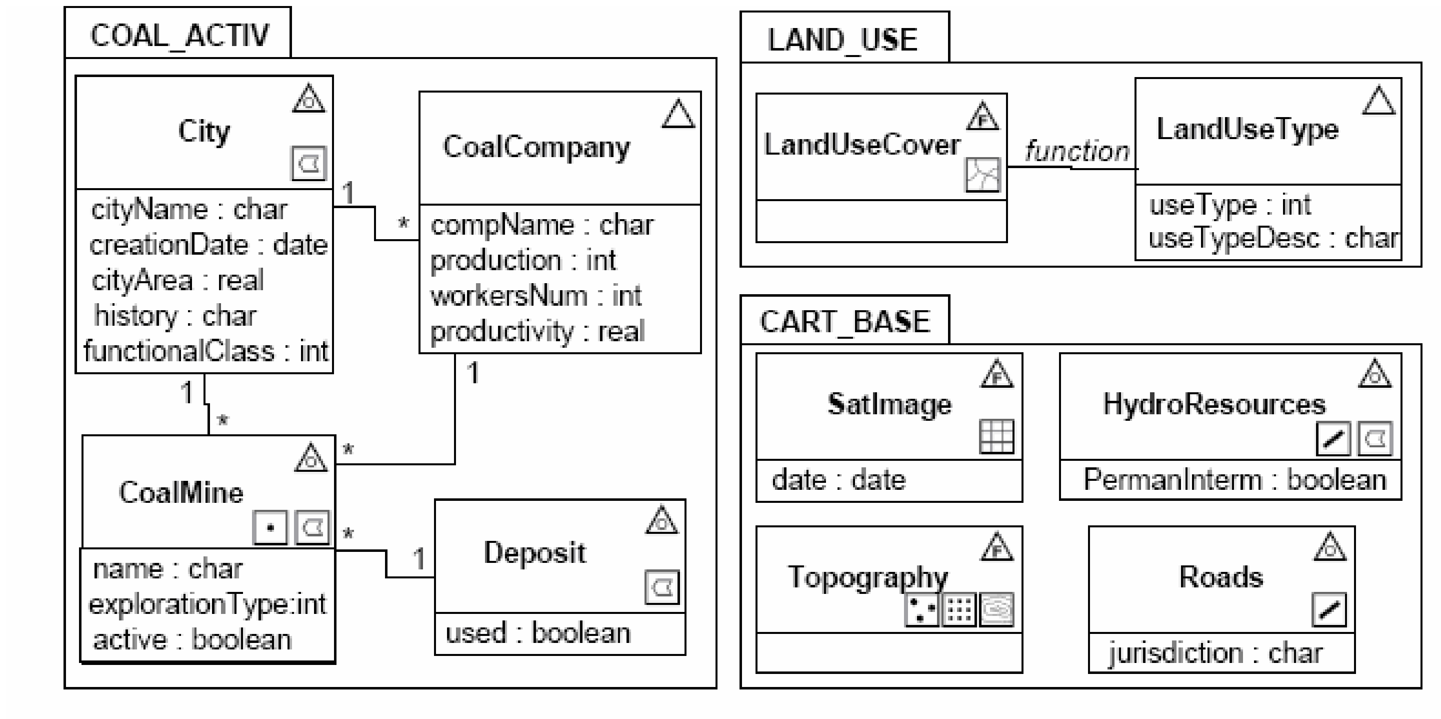
Stereotypes,

UML-GeoFrame Model

Packages and Stereotypes

- The use of packages and stereotypes greatly reduce the clutter on the diagram. Rather than including complete class hierarchies stereotypes are used.

Packages and Stereotypes



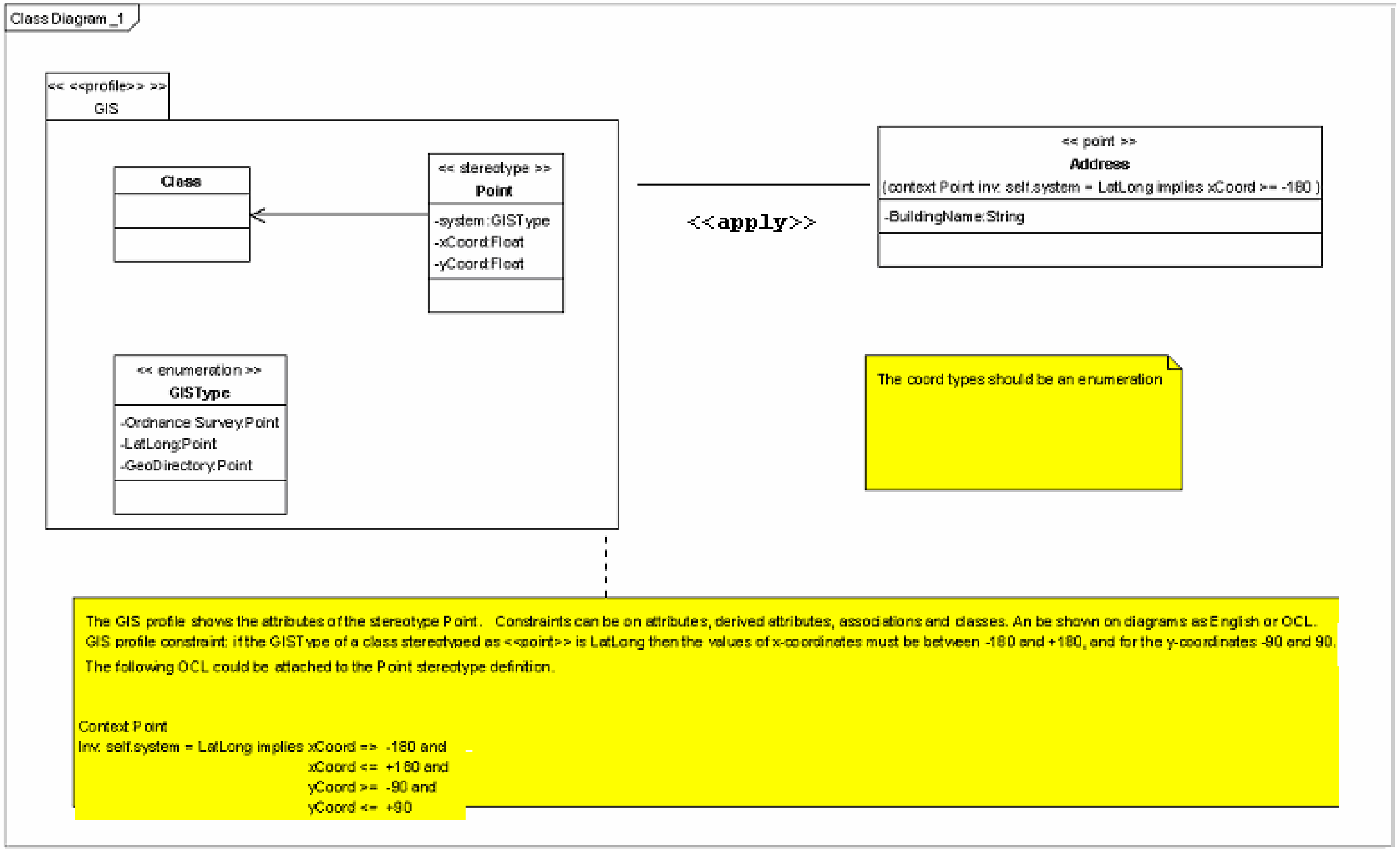
Stereotypes of the UML-GeoFrame Model

Mr. Patrick Browne, School of Computing
Dublin Institute of Technology

Profiles

- A profile is an extension to the core UML that keeps the UML meta-model intact, in other words it obeys the rules of UML. A UML profile is a subset of UML that is necessary and sufficient for a given development effort (e.g. Testing, WAP, or GIS). UML profiles are defined in the UML infrastructure volume of UML 2.0. Profiles are packages that are linked to other packages in a model by using the `<<apply>>` dependency. Model elements in the model package can then use the features that are in the profile, typically stereotypes. Profiles are supported in Rational Rose but not ArgoUML.
- The profile concept is similar to CafeOBJ module composition but without the formal underpinning of Category Theory.

Profiles (complex)



{context Point inv: self.system = LatLong implies xCoord >= -180 }

The coord types should be an enumeration

The GIS profile shows the attributes of the stereotype Point. Constraints can be on attributes, derived attributes, associations and classes. An be shown on diagrams as English or OCL.
GIS profile constraint: if the GISType of a class stereotyped as <<point>> is LatLong then the values of x-coordinates must be between -180 and +180, and for the y-coordinates -90 and 90.
The following OCL could be attached to the Point stereotype definition.

```
Context Point
Inv: self.system = LatLong implies xCoord => -180 and
                                   xCoord <= +180 and
                                   yCoord >= -90 and
                                   yCoord <= +90
```

Spatial Relations

- The UML-GeoFrame does not include spatial relation (e.g. contains, crosses). Filho et al feel that spatial relationships add unnecessary complexity to class diagram making it difficult for the user and developer to understand. Also, if they are formalized at the conceptual level then they must represent a complete and accurate set of relationships, this is not an easy task. The number of possible topological relationships can become very high. There is the difficulty of deriving meaningful names for these relationships. Also it may be hard for users to remember the appropriate usage.

Spatial Relations

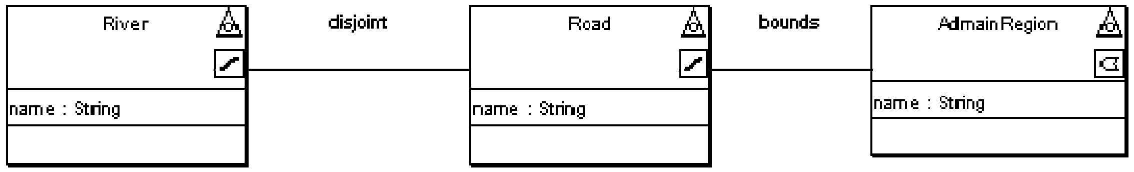
- Consider the Table on the next slide from Clementini et al. The authors describe spatial relations such as *"touch"* and *"overlap"* between points, lines and areas. By enumerating all possible combinations and removing those which cannot be drawn in space there are still 52 relations!

Spatial Relations

Relation Between	Number Possible	Actual Number
area/area	24	9
line/area	24	17
point/area	4	3
line/line	24	18
point/line	4	3
point/point	2	2
TOTALS =	82	52

Spatial Relations

- The stereotypes below from the `UML-GeoFrame` are used to indicate line and polygon. The user needs to be able to map these high level concepts to lines and polygons as implemented by the target system (e.g. PostGIS).
- Assume the spatial relationships `bounds` and `disjoint` (not implemented in `ArgoCASEGEO`) have their intuitive meanings. The meaning of such a relationship is much more complex than the basic UML association. Standard associations cannot express conditional constraints which are needed to express `disjoint`. Hence to define topological constraints more precisely requires OCL (e.g. in USE) or algebraic expressions (in CafeOBJ). The precise meaning of the `bounds` relationship depends on UML-GeoFrame relationship between lines and polygons.



Spatial Relations

- Spatial objects have complex relations which present a challenge for the basic UML association. Fiis-Christensen et al summarized the following relationships from the GIS literature:

Spatial Relations

Topological relationships	Binary topological relationships include contains, overlaps, passes trough, and touches
Metric relationships	Metric relationships involve distance and depend on the absolute positions of objects relative to a given reference system
Semantic relationships	These are relationships among objects, relevant at the conceptual level, that are neither topological nor metric. It could for example be that all land parcels have road access, either directly or through an allowed access via a neighbour parcel. When representing the objects in less expressive models, some semantic relationship may be represented as a topological relationship
Part-of relationships.	An object can consist of other objects. An example is a county consisting of municipalities.
Relationship constraints.	The relationship constraints are very important and are highly dependent on the type of relation between objects. A constraint can be topological: for example, it is not allowed to place a building in a lake. It is also important to be able to state exceptions. In the above example, it is may be allowed to build a house on pillars in a lake.

Spatial Relationships

- Recall, UML-GeoFrame is the spatial model used in ArgoCASEGEO. UML-GeoFrame models “semantic relations” which are implemented as UML associations, aggregations and compositions. For example, a building block could contain several parcels, but what this might mean in terms of spatial relations is not required in the conceptual model. UML-GeoFrame can express multiple representations, a field view, an object view.

CafeOBJ

- CafeOBJ is a much more expressive language than OCL, it is both a specification language and a functional programming language. This provides a “continuum of specification”. Models are represented by sorts (e.g. sets) and operations (e.g. functions) on those sorts. The operations are represented as mathematical equations. There are also conditional equations which will only be evaluated if the condition is satisfied. The basic CafeOBJ module has the following form:

```
module <ModId> {
[ <SortId> ]                                 --  sort declaration
op <OpForm> : <SortList> -> <Sort>           --  operation declaration
var <VarId> : <Sort>                         --  variable declaration
eq <Term> = <Term> .                         --  ordinary equation
ceq <Term> = <Term> if <Condition> .         --  conditional equation
}
```

- Reduction , or executions, can be performed in one of several logics. For example, the reduce command evaluates a term in equational logic.
- `reduce [ in <ModExp> :] <Term>`

CafeOBJ

- A module expression allows a modules to be composed in various ways. Invariants pre-conditions and post-conditions can be expressed as equations. CafeOBJ uses a variety logics, which allows us to model systems in different ways.
- On the next slide is the CafeOBJ specification in Figure 1(a).
- We show the Line-Point association is implemented in CafeOBJ.
- We show the how the CafeOBJ specification is executable.
- We show how CafeOBJ can be used as a simple functional language for prototyping.
- We show some queries
- Finally we will show how operations can change the state of an object.

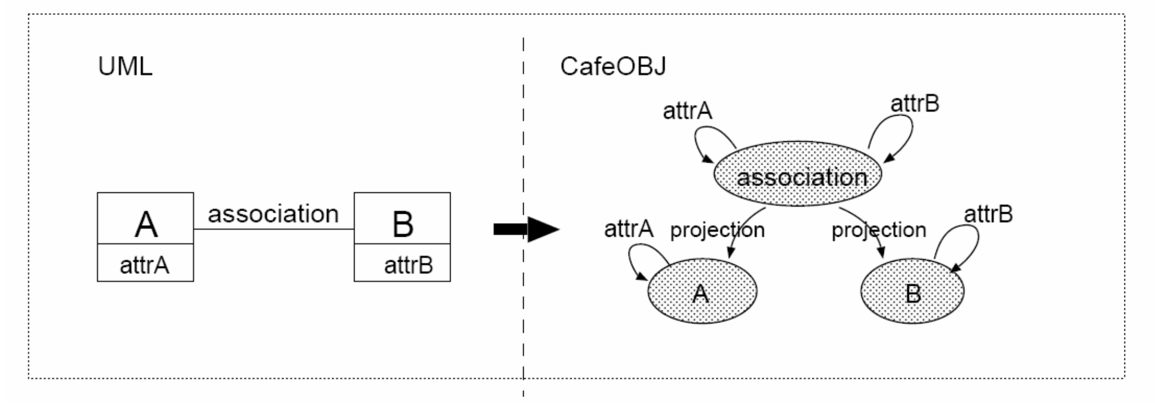
CafeOBJ

CafeOBJ modules	Associations

mod! POINT {
[Point]
pr(INT)
op x : -> Int
op y : -> Int
}
mod* LINE {
[Line]
}
mod* PATH {
[Path]
}
mod! MAKE-MAP {
pr(PATH-LINE)
pr(FOPL-CLAUSE)
ops p1 p2 p3 : -> Point
ops l1 l2 : -> Line
op path1 : -> Path
eq start(l1) = p1 .
eq finish(l1) = p2 .
eq start(l2) = p2 .
eq finish(l2) = p3 .
eq connectedPoints(p1 , p2)
     = l1 .
eq connectedPoints(p2 , p3)
     = l2 .
eq connectedLines(l1 , l2)
     = path1 .
}
	mod* LINE-POINT {
	pr(POINT)
	pr(LINE)
	op start : Line -> Point
	op finish : Line -> Point
	op connectedPoints : Point Point -> Line
	vars l1 l2 : Line
	vars p1 p2 : Point
	ceq connectedPoints(p1, p2) = l1 if p1 =/= p2 and finish(l1) == p2 and finish(l1) == p2 .
	}
	mod* PATH-LINE {
	pr(LINE-POINT)
	pr(PATH)
	op _|_ : Line Line -> Path
	op connectedLines : Line Line -> Path
	vars l1 l2 : Line
	var path1 : Path
	ceq connectedLines(l1,l2) = path1 if (start(l2) == finish(l1)) .
	}

Associations in CafeOBJ.

The formal methods community have used algebraic techniques to formalise parts of the UML. We use associations from Munakata and Kokichi. Their basic approach is shown in the Figure below. It is a bit like an association class.



CafeOBJ

- Like OCL, CafeOBJ can be used to check or test models. Expressions from executable equations can be evaluated using equational logic. The following expression should evaluate to true

```
red finish(l1) == start(l2)
```

- CaféOBJ can be used as a functional programming language. For example, the distance function can be defined as:
- `eq d( x1 , y1, x2 ,y2) = sqrt( (( x1 - x2) * ( x1 - x2) ) + (( y1 - y2 ) * ( y1 - y2)) ) .`
- Then evaluating
- `d(20.0,20.0,10.0,10.0) gives 14.142135623730951d0 : Float`
- In principle such simple functions can be programmed in OCL but OCL does not currently have a square root function.

Axioms of Metric Spaces

- A metric space has a function distance between the two points. For A, B, C elements of X.

The distance is 0 if and only if the points coincide
d(A, B) = 0 if and only if A = B.

The distance from A to B is the same as the distance from B to A
d(A, B) = d(B, A)

the sum of two sides of a triangle is never less than the third side.
d(A,B)+d(B,C) >= d(A,C)

- Numbers, strings, metric spaces, sets, sequences, partial orders and many mathematic frameworks can be encoded in CafeOBJ. We can thus reuse mathematical concepts.

CafeOBJ Metric Spaces

```
mod* METRIC1 {
pr (R2)
op d : Vect Vect -> Float
vars vx vy zx zy wx wy : Float
vars p1 p2 p3 : Vect
--  symmetry
eq d(p1 , p2) = d( p2 , p1 ) .
eq d(p1 ,p2) >= (0.0) = true .
--  reflexive
ceq d( p1, p2 ) = (0.0) if p1 == p2 .
--  triangle inequality
eq d(p1 , p2) + d(p2 , p3) >= d(p1 , p3) = true .

-- d(vx,vy ) > 0 if (vx =/= vy ) and d(vx,vx)=0 (non
   negativity)
}
```

Querying in CafeOBJ

- The query below asks does there exist a point, P1, that starts l1 in Figure 1a.

```
goal \E[P1:Point] connectedPoints(P1,p2) == l1 .
```

- In this query , the logical variable P1 will be instantiated with a constant if it can be found. In this case CafeOBJ will correctly find p1 (note lower case).
- The important point about this example is that it permits a form of querying that can be applied to basic equations. It also extends the basic theorem proving that is available in CafeOBJ beyond that possible with pure equational logic (equational logic uses substitution of equals for equals rather than the *modus ponens* of FOPL as its main inference rule).

Object Orientation in CafeOBJ

- CafeOBJ has two ways of represent OO semantic, hidden algebras and rewrite logic. Rewrite logic uses ‘rules’ rather than pure equations. The following rule represents a state transition of a one dimensional point object P.
- `rl moveRight(P , Dist) <P:Point|x = N> => < P: Point | x = (N+Dist) > .`
- This is executed with `exec` command which evaluates the expression under rewrite logic.
- `exec moveRight(P , 10) < P : Point | x = 10 > .`
- the result
- `< P : Point | x = 20 > : Point`
- The position is appropriately updated. Further we can mix pure equations and rewrite rules and conditional rewrite rules. In CafeOBJ rewrite logic can be used for *algebraic refinement*, which is supported by proof technologies. This idea is not found in UML/OCL.

CafeOBJ Specifications

- The denotation CafeOBJ specification represents a class of models or possible implementations of the specification.
- Specifications are formal description of certain class of models which exist as abstract mathematical object.
- A specification can have one model (initial) are several (loose).

Signatures, Sentences, Models

$$\begin{tikzcd}
 & \mathbf{Set} \\
\mathbf{Sign} \arrow[ur, "Sen"] \arrow[dr, "\mathbf{Mod}"'] & \models \\
 & \mathbf{Cat}^{op}
\end{tikzcd}$$

$$\begin{tikzcd}
\Sigma \arrow[d, "\phi"'] & \mathbf{Mod}(\Sigma) & \models_{\Sigma} & Sen(\Sigma) \arrow[d, "Sen(\phi)"] \\
\Sigma' & \mathbf{Mod}(\Sigma') \arrow[u, "\mathbf{Mod}(\phi)"] & \models_{\Sigma'} & Sen(\Sigma')
\end{tikzcd}$$

CafeOBJ Specifications

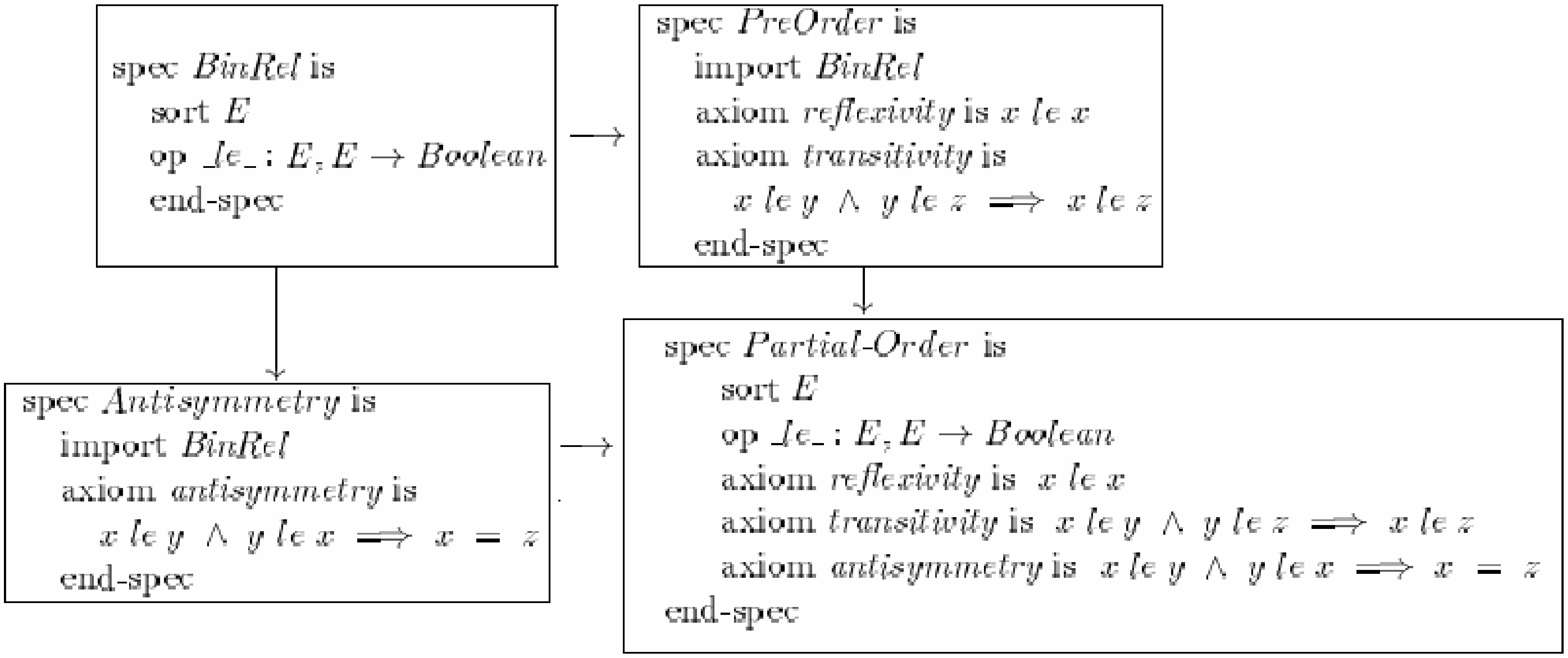
- The previous shows that a ‘signature morphism’, where a signature is mapped to another signature or renamed.
- The CafeOBJ uses module expressions to implement the concept of signature and sentence translation.

CafeOBJ module expression

- One or more imports
- Parameterized modules
- Translation
- Sums, and
- Views and parameter instantiation.

Algebraic module expressions

Constructing theories and models from components



```
spec BinRel is
    sort E
    op _le_ : E, E → Boolean
    end-spec
```

```
spec PreOrder is
    import BinRel
    axiom reflexivity is x le x
    axiom transitivity is
        x le y ∧ y le z ⟹ x le z
    end-spec
```

```
spec Antisymmetry is
    import BinRel
    axiom antisymmetry is
        x le y ∧ y le x ⟹ x = z
    end-spec
```

```
spec Partial-Order is
    sort E
    op _le_ : E, E → Boolean
    axiom reflexivity is x le x
    axiom transitivity is x le y ∧ y le z ⟹ x le z
    axiom antisymmetry is x le y ∧ y le x ⟹ x = z
end-spec
```

Mr. Patrick Browne, School of Computing
Dublin Institute of Technology

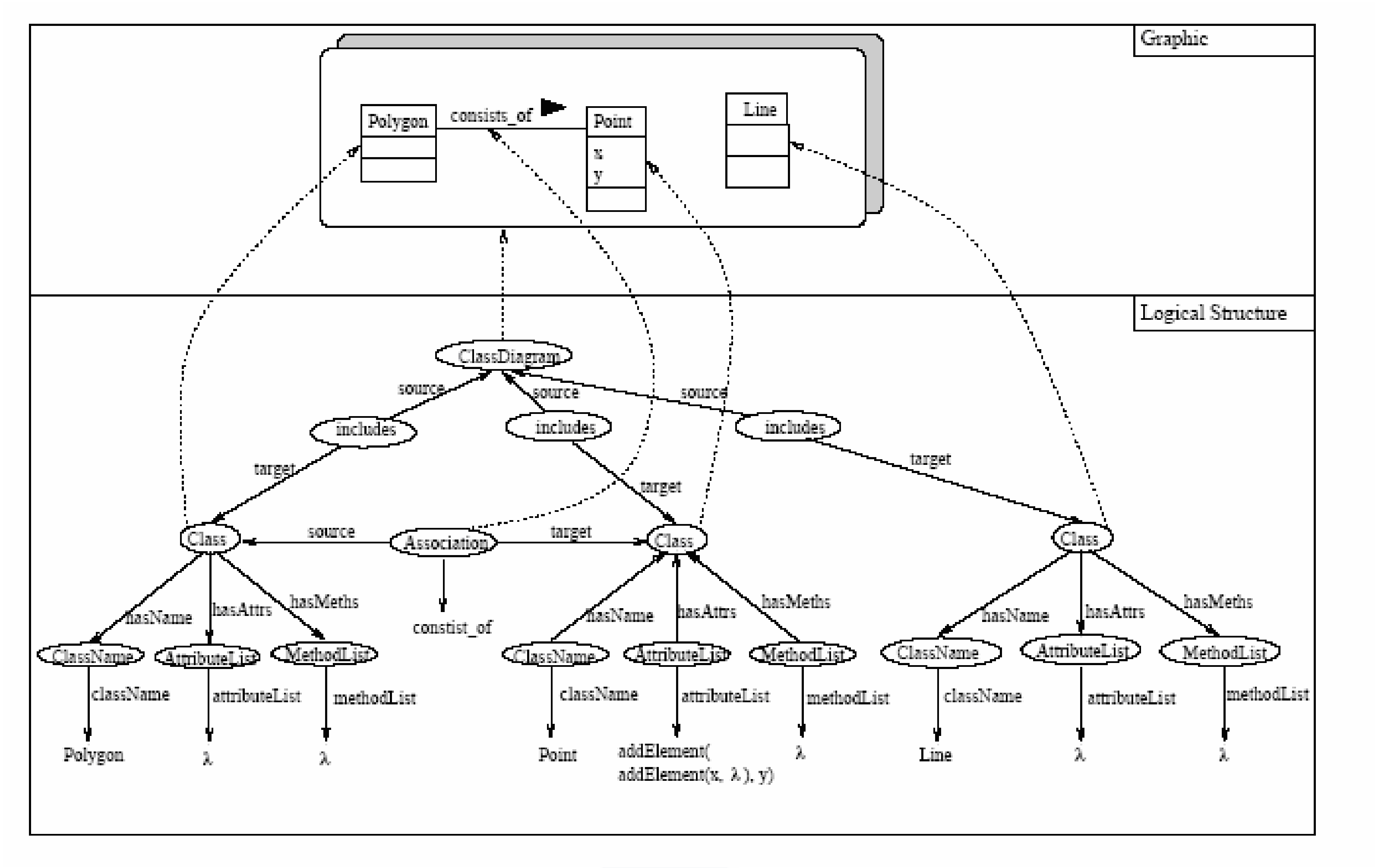
Summary of CafeOBJ

- An executable specification language based on logic.
- Amenable to various types of theorem proving, proof checking, and model checking.
- Powerful module system

UML v Algebraic specifications

- The UML supplies a fair amount of implicit knowledge, which has to be explicitly represented in an algebraic specification. This point is illustrated in the next slide which shows how a UML diagram might be formalized using an algebraic specification (Bardohl and Taentzer 1997).

UML v Algebraic



Algebraic Specifications

- Tools like CafeOBJ are often considered excessively formal and difficult to use. However we feel:
- In GIS traditionally the core formalisms were analytical geometry, and the closely relate measurement sciences of surveying and geodesy. The presentation of the data was handled by cartography which has a strong basis in more subtle cognitive sciences. Though we do not discuss it here algebraic approaches have strong links to both mathematic and cognitive science.
- One big advantage of CafeOBJ is that specifications can be <u>composed</u> preserving valid axioms. This is difficult with most UML tools.

Commercial System

- ESRI’s users can model geo-data with Rational Rose and Visio. These are fairly well integrated with ESRI’s products.
- Visio uses templates to represent the ArcInfo UML Model. This permits a fairly easy transition between UML the GIS. Their object model has five packages:
- Logical View
- ESRI Classes
- ESRI Interfaces
- ESRI Network
- Workspace

Interoperability between design tools and other systems.

- By either open source or commercial standards algebraic tools like CafeOBJ do not integrate with other software such as databases, programming languages and other CASE tools. This means it would be difficult to import spatial model libraries (if they existed!). However, they do integrate with the conceptual worlds of mathematics and logic. At this stage they should be considered as research tools and to be used in applications where rigor and precision are required (e.g. standards). They can also be used to help define desirable criteria for spatial specification languages themselves.

Interoperability between design tools and other systems.

- The situation is a little better in the UML/OCL approach. Notwithstanding the use of XMI for model interchange it can still be difficult to move models between systems. We may need to move models between CASE tool or conceptual spatial frameworks. For example, Perceptory (Bédard, 2006) is a free spatial-temporal modeling tool, it is based on the proprietary MS Visio. It has UML profiles for ISO-TC211 19110 (contents cataloguing) and ISO-TC211 19115 (meta-data) spatial standards. Using ESRI's XMI exporter it should be possible to export these profiles to ArgoUML. Open source tools tend to be more difficult use with less point and click and more manual bashing, code hacking and editing of configuration files.

Conclusion

- It would be nice if there was a single specification language that would be the optimum choice for every possible use. Unfortunately, the reality is that many languages are necessary because of the fragmented nature of computing (e.g. various paradigms such as OO) and different application areas. We have show two possible approaches to modeling the geo-domain, namely the UML/OCL approach and an algebraic approach using CafeOBJ. The UML/OCL approach is well established in the geo-domain and the IT community in general.

Conclusion

- OCL provides a semi-formal language in which many useful constraints and queries can be expressed. ArgoCASEGEO provides a good set of geographic extensions to assist the GIScientist. Although this approach lacks strict formality the graphic capability make it very useful for initial requirement gathering. However, it has too many limitations for large scale applications such as the specification of the OGC standard. There is a general move to formalized the UML and make it more precise. Languages such as CafeOBJ already have this precision.

Conclusion

- There are issues between models and systems. For example a user would find it difficult to use an ISO spatial standard if the proposed GIS does not support it. ArgoCASOGEO allows users to use the UML-GeoFrame which can be converted to the more implementation oriented ESRI shape file format. Other tools such as Perceptory can outputs data that conforms to ISO models or Oracle tables.

Conclusion

- We feel that the main problems with the algebraic approach are lack of graphics, they are difficult for users or customers to understand, and lack of libraries for geo-specification (but some are available in CASL).
- We feel that both algebraic and OO approaches are useful at different stages and for different types of project.
- The UML/OCL approach is fairly well integrated into the general software environment but poorly integrated into the mathematical world.
- Algebraic tools are ideal for use in standards work and constructing large system. In both cases there is a need for precise mathematical modeling that can scale up. Once such specifications or systems are build less technical users can use the pre-constructed models for their own application.

Conclusion

- We did not contrast the relative value of proofs and testing.
- We hope to extend ArgoUML and USE to include a precise definition of OGC's SFSS.
- In order to cognitively engage in the design we feel that designers need to be proficient in formal and semi-formal specifications. The semi-formal approach can be used when a certain looseness is acceptable and the formal approach is need when precision and proof is needed.

Conclusion

- Problems syntactic and semantic gaps:
- UML <–> Spatial/temporal concepts
- UML <–> CafeOBJ
- CafeOBJ <–> Spatial/temporal concepts
- Cognitive Theories <–> Software realization
- Gaps in specifications can lead to interoperability problems.
- In theory these gaps can be bridged by truth preserving morphisms found in the algebraic approach.